下载APP

05 | The Google File System （三）： 多写几次也没关系

2021-09-29 徐文浩

《大数据经典论文解读》 课程介绍 >

讲述：徐文浩

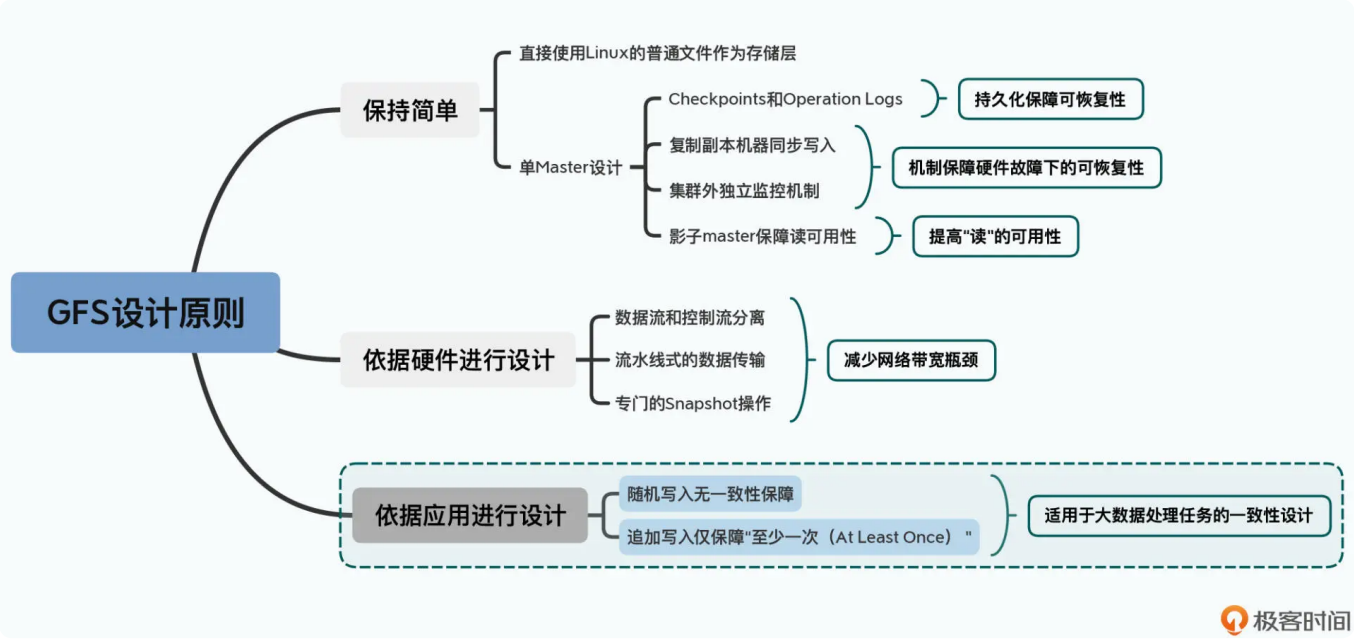
时长 15:32 大小 14.23M

你好，我是徐文浩。在前面的两讲中，我们一起探讨了 GFS 系统设计中秉持的两个原则，分别是“保持简单”和“根据硬件特性设计系统”，而今天我们要讨论的 GFS 的最后一个设计特点，是“**放宽数据一致性的要求**”。

分布式系统的一致性要求是一个很有挑战的话题。如果说分布式系统天生通过更多的服务器提升了性能，是一个天然的优点，那么在这些通过网络连起来的服务器之间保持数据一致，就是一个巨大的挑战。毕竟网络传输总有延时，硬件总会有故障，你的程序稍有不慎，就会遇到甲服务器觉得你的钱转账失败，而乙服务器却觉得钱已经转走了的情况。可以说，**一致性是分布式系统里的一个永恒的话题**。

不过 2003 年的 GFS，对于一致性的要求，是非常宽松的。一方面，这是为了遵循第一个设计原则，就是“保持简单”，简单的设计使得做到很强的一致性变得困难。另一方面，

则是要考虑“硬件特性”，GFS 希望在机械硬盘上尽量有比较高的写入性能，所以它只对顺序写入考虑了一致性，这就自然带来了宽松的一致性。



好，废话不多说，让我们切入正题。

随机写入只是“确定”的

通过上一讲的学习，我们知道了在 GFS 中，客户端是怎么把数据写入到文件系统里的。不过，我们并没有探讨一个非常重要的问题，就是数据写入的一致性（Consistency）问题。这个一致性，也是我们常常听说的 CAP 理论中的 C，即一致性、可用性、分区容错性在分布式系统中，三者不能兼得中的一致性问题。这个 C，也正来自于一致性的英文 Consitency 的首字母。

我们先来看看，一致性到底指的是什么东西。在 GFS 里面，主要定义了对一致性的两个层级的概念：

> 第一个，就叫做“一致的（Consistent）”。这个就是指，多个客户端无论是从主副本读取数据，还是从次副本读取数据，**读到的数据都是一样的**。
>
> 第二个，叫做“确定的（Defined）”。这个要求会高一些，指的是对于客户端写入到 GFS 的数据，能够**完整地被读到**。可能看到这个定义，你还是不清楚，没关系，我下面会给你详细讲解“确定的”到底是个什么问题。

	写入	记录追加
顺序写入成功	确定的	确定的，不过夹杂着不确定的数据
并发写入成功	一致的但是非确定的	
写入失败	不一致	

极客时间

GFS论文中的Table 1：在变化后文件区域的状态

现在，我们来看看 GFS 的论文里留的一个表格，其中告诉了我们 GFS 对于数据写入的一致性问题。

首先，如果数据写入失败，GFS 里的数据就是不一致的。

这个很容易理解，GFS 里面的数据写入，并不是一个事务。上一讲里说过，主副本会把写入指令下发到两个次副本，如果次副本写入失败了，它会告诉主副本。但是，此时主副本和另一个次副本都已经写入成功了。那么这个时候，GFS 里的三个副本的数据，就是不一致的了。不同的客户端，就可能读到不同的数据。

其次，如果客户端的数据写入是顺序的，并且写入成功了，那么文件里面的内容就是确定的。

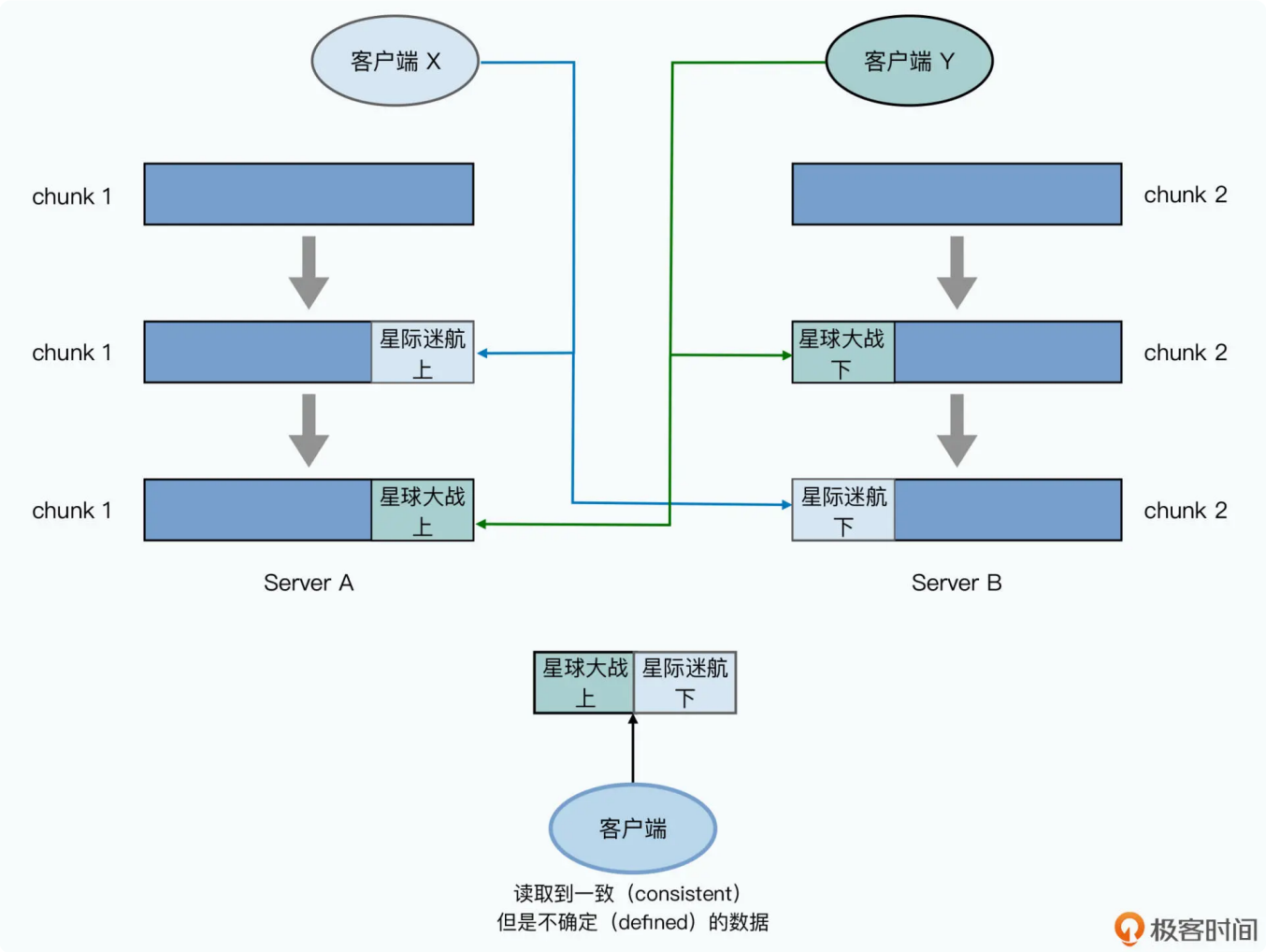
比如，你先往一个文件里，写入一部电影《星球大战》，这个时候，客户端无论从哪个副本读数据，读到的都是星球大战。然后再写入《星际迷航》，那么客户端再读数据，读到的也一定是《星际迷航》。

但是，如果由多个客户端并发写入数据，即使写入成功了，GFS 里的数据也可能会进入一个一致但是非确定的状态。

也就是说，两个客户端并发往一个文件里面写数据，一个想要写入《星球大战》，一个想要写入《星际迷航》，两个写入都成功了。这个时候，GFS 里面三份副本的数据是一样的，客户端读到的数据无论是从哪个副本里读，都是一样的。

但是呢，客户端可能读出来的数据里，前一小时是《星球大战》，后一小时是《星际迷航》。无论哪个时间节点去读数据，客户端都不能读到一部完整的《星球大战》，或者是

《星际迷航》。



那么，**为什么 GFS 的数据写入会出现一致但是非确定的状态呢？**这个来自于两个因素。

第一种因素是在 GFS 的数据读写中，为了减轻 Master 的负载，数据的写入顺序并不需要通过 Master 来进行协调，而是直接由存储了主副本的 chunkserver，来管理同一个 chunk 下数据写入的操作顺序。

第二种因素是随机的数据写入极有可能要跨越多个 chunk。

我们在写入《星球大战》和《星际迷航》的时候，前一个小时的电影是在 chunk 1，对应的主副本在 server A，后一个小时的电影是在 chunk 2，对应的主副本在 server B。然后写入请求到 server A 的时候，《星际迷航》在前，《星球大战》在后，那么《星球大战》的数据就覆盖了《星际迷航》。

而到 server B 的时候则是反过来，《星际迷航》又覆盖了《星球大战》。于是，就会出现客户端读数据，前半段是《星球大战》，后半段是《星际迷航》的奇怪现象了。

其实，这个一致但是非确定的状态，是因为随机的数据写入，没有**原子性**（Atomic）或者**事务性（**Transactional）。如果想要随机修改 GFS 上的数据，一般会建议使用方在客户端的应用层面，保障数据写入是顺序的，从而可以避免并发写入的出现。

追加写入的“至少一次”的保障

看到这里，你可能要纳闷了。不是说 GFS 支持几百个客户端并发读写吗？现在怎么又说，写入要客户端自己保障是“顺序”的呢？

实际上，这是因为随机写入并不是 GFS 设计的主要的数据写入模式，GFS 设计了一个专门的操作，叫做**记录追加**（Record Appends）。这是 GFS 希望我们主要使用的数据写入的方式，而且它是原子性（Atomic）的，能够做到在并发写入时候是基本确定的。

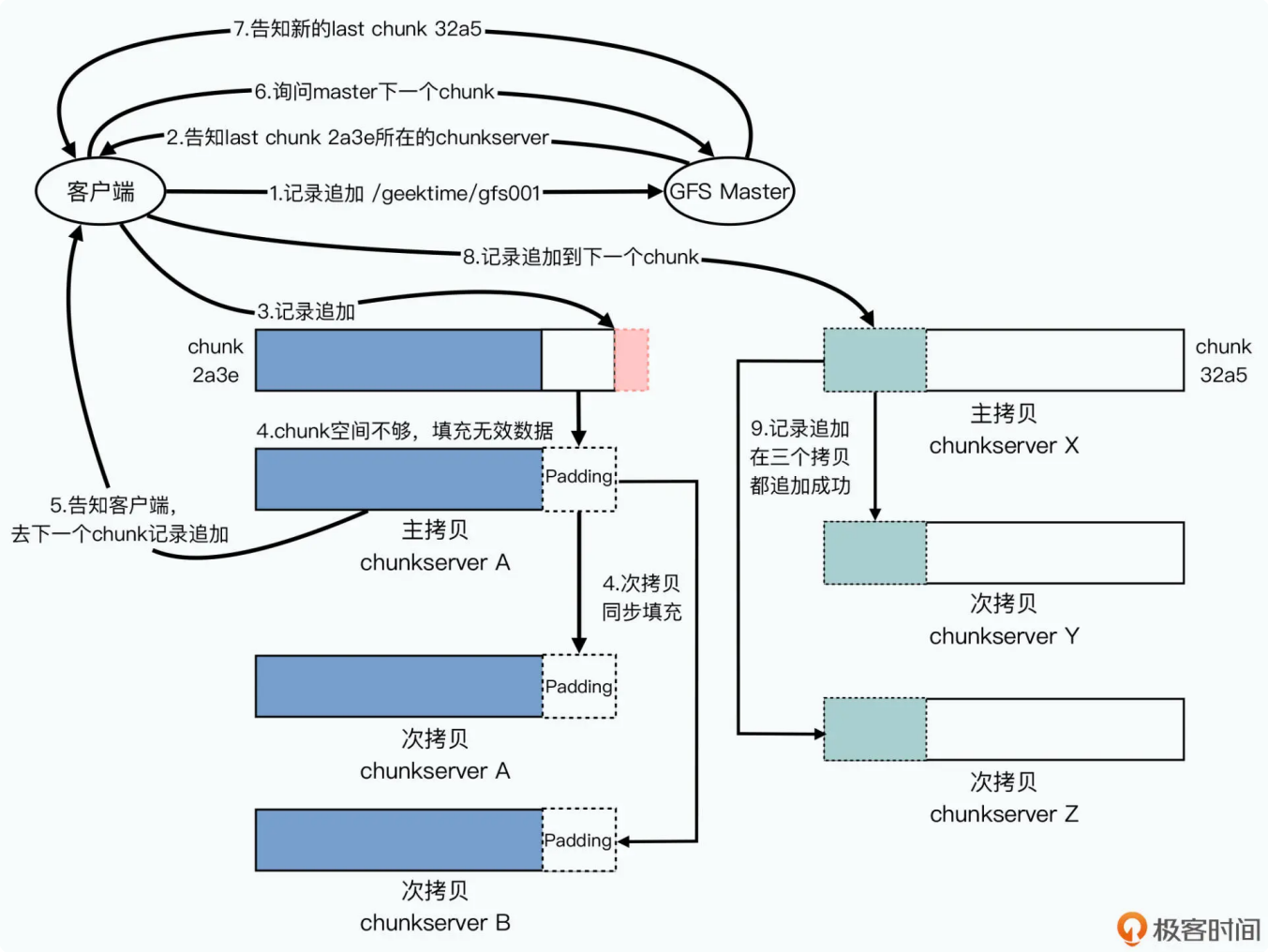
不过，作为一个严谨的工程师，听到“基本”这两个字你可能心里一颤。别慌，接下来我就带你一起看一下，这个基本到底是怎么回事儿。

GFS 的记录追加的写入过程，和上一讲的数据写入几乎一样。它们之间的差别主要在于，GFS 并不会指定在 chunk 的哪个位置上写入数据，而是告诉最后一个 chunk 所在的主副本服务器，“我”要进行记录追加。

这个时候，主副本所在的 chunkserver 会做这样几件事情：

1. 检查当前的 chunk 是不是可以写得下现在要追加的记录。如果写得下，那么就把当前的追加记录写进去，同时，这个数据写入也会发送给其他次副本，在次副本上也写一遍。
2. 如果当前 chunk 已经放不下了，那么它先会把当前 chunk 填满空数据，并且让次副本也一样填满空数据。然后，主副本会告诉客户端，让它在下一个 chunk 上重新试验。这时候，客户端就会去一个新的 chunk 所在的 chunkserver 进行记录追加。
3. 因为主副本所在的 chunkserver 控制了数据写入的操作顺序，并且数据只会往后追加，所以即使在有并发写入的情况下，请求也都会到主副本所在的同一个 chunkserver 上排队，也就不会有数据写入到同一块区域，覆盖掉已经被追加写入的数据的情况了。

4. 而为了保障 chunk 里能存的下需要追加的数据，GFS 限制了一次记录追加的数据量是 16MB，而 chunkserver 里的一个 chunk 的大小是 64MB。所以，在记录追加需要在 chunk 里填空数据的时候，最多也就是填入 16MB，也就是 chunkserver 的存储空间最多会浪费 1/4。



那么到了这里，你可能要问了：**如果在主副本上写入成功了，但是在次副本上写入失败了怎么办呢？这样不是还会出现数据不一致的情况吗？**

其实在这个时候，主副本会告诉客户端数据写入失败，然后让客户端重试。不过客户端发起的重试，并不是在原来的位置去写入数据，而是发起一个新的记录追加操作。这个时候，可能已经有其他的并发追加写入请求成功了，那么这次重试会写入到更后面。

我们可以一起来看这样一个例子：有三个客户端 X、Y、Z 并发向同一个文件进行记录追加，写入数据 A、B、C，对应的三个副本的 chunkserver 分别是 Q、P、R。

主副本先收到数据 A 的记录追加，在主副本和次副本上进行数据写入。在 A 写入的同时，B，C 的记录追加请求也来了，这个时候写入会并行进行，追加在 A 的后面。

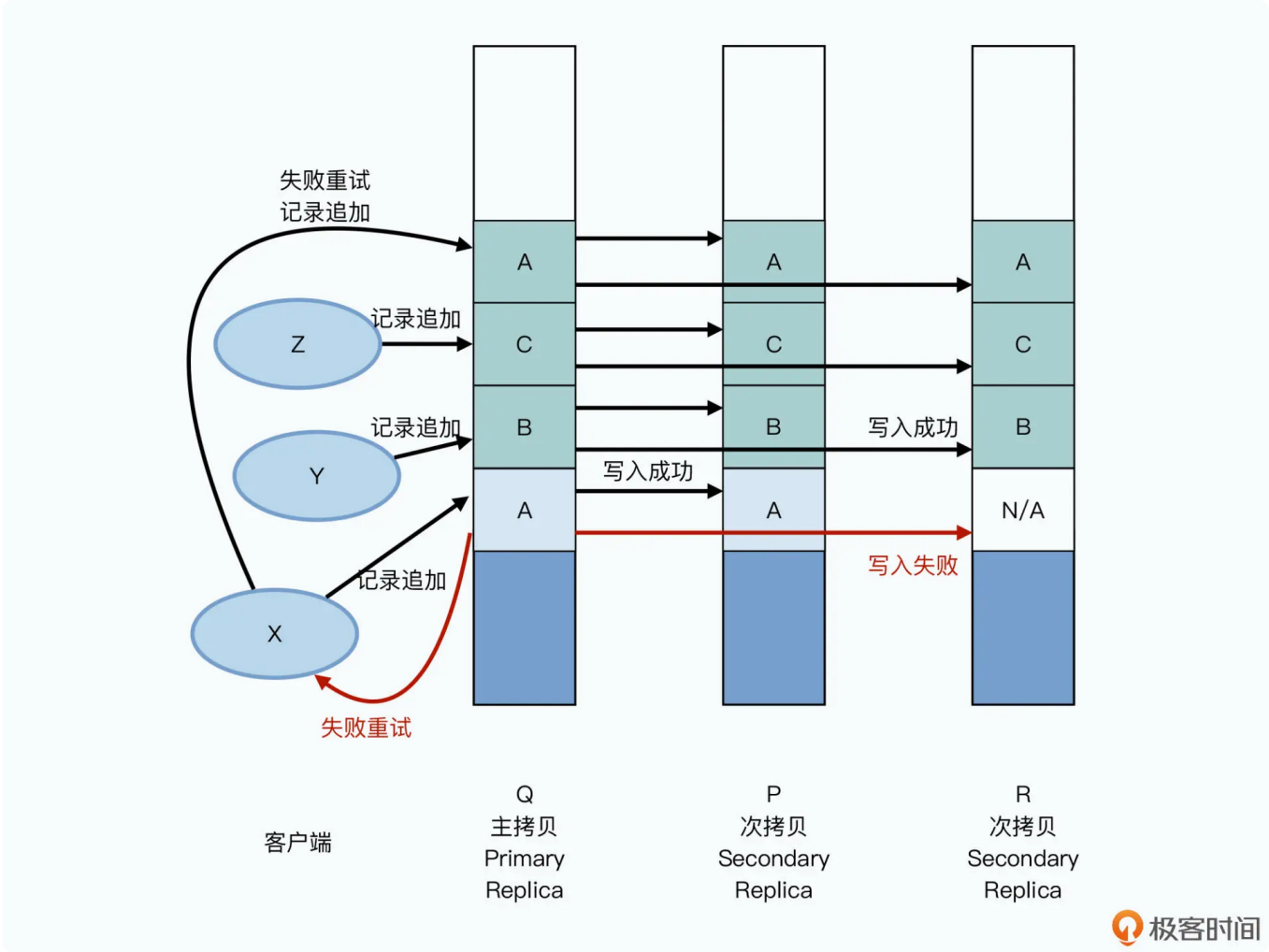
这个时候，A 的写入在某个次副本 R 上失败了，于是主副本告诉客户端去重试；同时，客户端再次发起记录追加的重试，这次的数据写入，不在 A 原来的位置，而会是在 C 后面。

如此一来，在 B 和 C 的写入，以及 A 的重试完成之后，我们可以看到：

在 Q 和 P 上，chunkserver 里面的数据顺序是 A-B-C-A；

但是在 R 上，chunkserver 里面的数据顺序是 N/A-B-C-A；

也就是 Q 和 P 上，A 的数据被写入了两次，而在 R 上，数据里面有一段是有不可用的脏数据。



所以在这个记录追加的场景下，GFS 承诺的一致性，叫做“**至少一次（At Least Once）**”。也就是写入一份数据 A，在重试的情况下，至少会完整地在三个副本的同一个位置写入一次。但是也可能会因为失败，在某些副本里面写入多次。那么，在不断追加数

据的情况下，你会看到大部分数据都是一致的，并且是确定的，但是整个文件中，会夹杂着少数不一致也不确定的数据。

解决一致性的机制

看到这里，你发现没有，GFS 的写入数据的一致性保障是相当低的。它只是保障了所有数据追加至少被写入一次，并且还保障不了数据追加的顺序。这使得客户端读取到的副本中，可能也会存在重复的数据或者空的填充数据，这样的文件系统实在不咋样。

不过，**这个“至少一次”的机制，其实很适合 Google 的应用场景**。你想像一下，如果你是一个搜索引擎，不断抓取网页然后存到 GFS 上。其实你并不会太在意这个网页信息是不是被重复存了两次，你也不太会在意不同的两个网页存储的顺序。而且即使你在意这两点，比如你存的不是网页，而是用户的搜索日志或广告展示和点击的日志数据。或者你担心数据写入失败，带来的是部分不完整的数据，也有很多简单的解决办法。

事实上，GFS 的客户端里面自带了对写入的数据去添加**校验和**（checksum），并在读取的时候计算来验证数据完整性的功能。而对于数据可能重复写入多次的问题，你也可以对每一条要写入的数据生成一个唯一的 ID，并且在里面带上当时的时间戳。这样，即使这些日志顺序不对、有重复，你也可以很容易地在你后续的数据处理程序中，通过这个 ID 进行排序和去重。

而这个“至少一次”的写入模型也带来了两个巨大的好处。

第一是**高并发和高性能**，这个设计使得我们可以有很多个客户端并发向同一个 GFS 上的文件进行追加写入，而高性能本身也是我们需要分布式系统的起点。

第二是**简单**，GFS 采用了一个非常简单的单 master server，多个 chunkserver 架构，所有的协调动作都由 master 来做，而不需要复杂的一致性模型。毕竟，2003 年我们只有极其难以读懂的 Paxos 论文，Raft 这样的分布式共识算法要在 10 年之后的 2013 年才会诞生。而简单的架构设计，使得系统不容易出 Bug，出了各种 Bug 也更容易维护。

而即使 GFS 里的数据随机写入能够保障**确定性**，在那个年代的实用价值也不高。因为那个时候，大家用的还都是 5400 转，或者 7200 转的机械硬盘。如果你读过我的《深入浅出计算机组成原理》，你一定还记得机械硬盘的 IOPS 也就是在 70 左右，几百个客户端要是并发写入一个位置，硬盘根本抗不住。

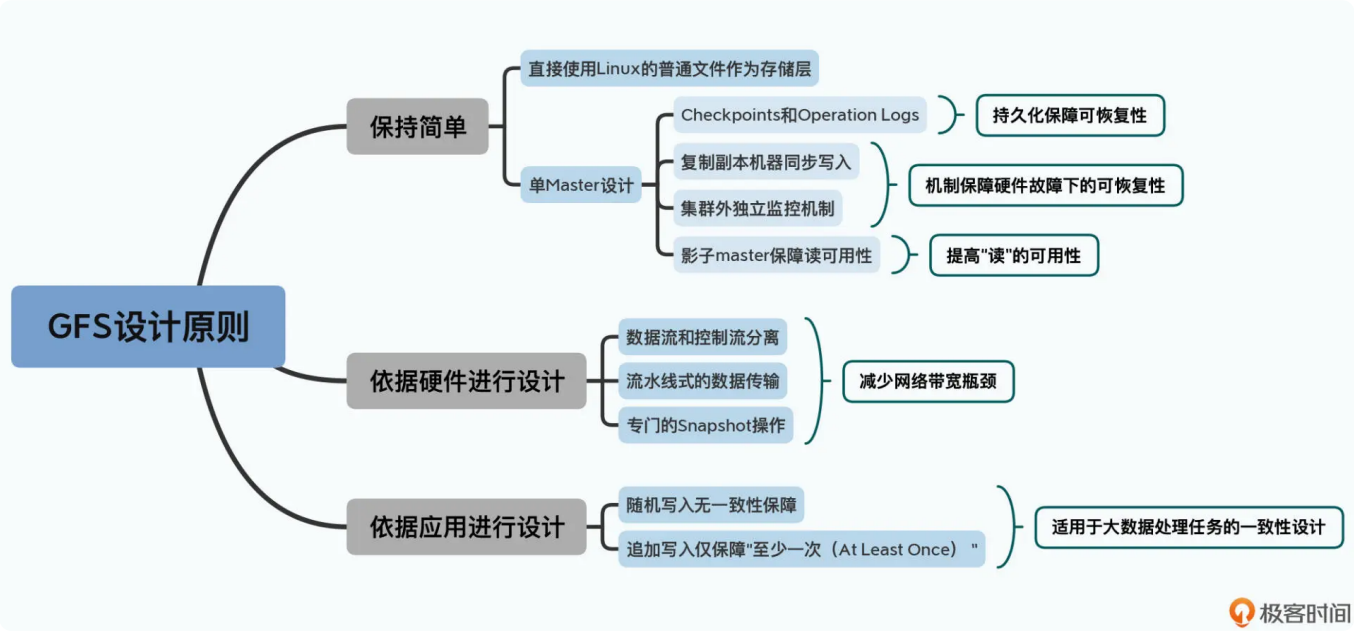
对于随机数据写入的一致性，我会在后面 Bigtable 的论文里，再带你看看 Google 是如何在这个不可靠的 GFS 上，架起一个高可用性、可以随机读写的 KV 数据库。

小结

好了，到这里，我们通过三讲把 GFS 的论文讲完了。在今天的最后一讲中，我们深入剖析了 GFS 非常宽松的一致性模型。对于随机数据写入，GFS 针对成功的并发写入，**只保障一致性，但是不保障确定性**。

而对于记录追加这样专门设计出来的操作，GFS 也**只保障“至少一次”的写入**，避免不了确定的数据中，夹杂着不一致和不确定的脏数据。而这些数据，都需要你通过校验和乃至应用层自行去重来进行处理。这些设计和机制，都是和 GFS 面临的应用场景所匹配的，即高并发大量追加写入新的日志、网页、生成的索引等等的应用场景。

回顾过去三讲，我们可以看到 GFS 的设计原则，就是简单、围绕硬件性能设计，以及在这两个前提下对于一致性的宽松要求。可以说，GFS 不是一个“黑科技”系统，而是**一个非常优秀的工程化系统**。在后面的 MapReduce 以及 Bigtable 的论文中，我们还会反复看到类似的设计选择。



推荐阅读

今天我们所讲的，其实是在大数据及分布式领域中一个非常重要的主题，那就是 CAP 问题。在之后的很多论文里，你也可以看到不同系统是如何处理 CAP 问题带来的挑战的。在学完今天这节课之后，我比较推荐你去读一读 2012 年，在 CAP 理论发表 12 年后，作者

埃里克·布鲁尔（Eric Brewer）对 CAP 问题的回顾文章《CAP 理论十二年回顾：“规则”变了》。

思考题

最后，我们还是来针对今天所学的内容，来思考一个简单的问题：GFS 的文件，只能保障“至少一次”的追加写入。那么，如果我们通过 GFS 的客户端要写入一部电影到 GFS，然后过一阵再读出来，我们都可以有哪些方式，来保障这个电影读取之后能够正常播放呢？

无论是在读、还是写、还是播放的时候进行技术处理的方法，你都可以在留言区提出来，我们一起讨论。如果觉得有收获，欢迎你把今天的内容分享给更多的朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 3 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | The Google File System （二）： 如何应对网络瓶颈？

下一篇 06 | MapReduce（一）：源起Unix的设计思想

精选留言 (5)

写留言

吴小智
2021-10-01

老师能否抽时间讲一下 GFS 中 5.2 Data Integrity 章节关于数据完整性的内容，论文说是不同的 chunkserver 独立检查数据的完整性，不怎么看懂，老师能否解答下。

作者回复: 对于数据完整性和校验的基本问题，可以去看「深入浅出计算机组成原理」里面的49和50两讲。

关于GFS我简单回单一下：

GFS其实包括后面的BigTable这些数据都是用类似的方式来保障 Data Integrity。

* 把数据按照固定尺寸打包，比如GFS里就是64K大小一个包
* 针对这个包计算一个Checksum验证码，通常用CRC33
* 然后读的时候会校验，发现数据和校验码不一致，就认为Data Corrupted，然后从别的副本拿到正确的数据

这个事情chunkserver不需要任何额外信息，只需要本地的数据就好了，所以叫独立检查数据的完整性。

3

在路上
2021-09-29

徐老师好，非常感谢对CAP问题的分享，希望能多分享一些中文资料。回答老师的问题，如果保证一部电影写入GFS之后，读出来能正常播放？可以将电影数据拆分成k份，每一份的大小小于GFS单次追加的数据，在每一份数据前面增加电影编号和数据块编号，以便在读取时重新组装数据。如果数据顺序写入GFS，读取时只需要去重，如果数据并发写入GFS，读取时既需要去重，也需要重排序。考虑到播放电影这个业务场景，可以采用64位数...
展开

作者回复: 在读取数据的时候排序不是一个好办法，主要是电影这么大的数据肯定是流式传输获取，不太可能全都读过来排序。

不过分组组装的思路是没错的。

1

zhanyd
2021-09-29

如果我们通过 GFS 的客户端要写入一部电影到 GFS，然后过一阵再读出来，我们都可以有哪些方式，来保障这个电影读取之后能够正常播放呢？

可以按照老师说的，给每一条要写入的数据生成一个唯一的ID，并且在里面带上当时的时间戳，读取的时候再按ID的顺序去重读取即可。
展开

作者回复: 这份分组传输的方式的确是一个有效的办法

1

峰

2021-09-29

如果在写入时就保证文件是ok的，那么就是要做个失败重试，即，写入失败就整个放弃重新写一个文件。
另一个思路是，记录下每次发其写入请求是失败还是成功，以及每次写入的大小，也就形成了一个list（成功失败，写入大小）每次读电影文件的时候，就可以根据这个信息，读文件了，比如开始读失败了，所以跳过这次写入大小，读成功了，则读取这次写入大小。 ...
展开

作者回复:
这是一个合理的思路，就是在写的时候保障，并且通过元数据管理来做这个事情。这样的话，也能够分段并行写。

1

webmin

2021-10-01

一个chunk有64MB，追加一次16MB，这个是不是有点像是一个嵌套数组，第一维定位chunk，第二维定位每次追加写入，管理文件块与管理内存块比较类似。
思考题：
可以参考TCP协议，TCP是流式协议，向网络写入1MB，在传输的过程中，这一1MB会被切分为若干包，在包外套上一层TCP协议的头，其中有Seq等信息，交换机在传输的过程中...
展开

作者回复: 不太一样，追加一次不是一定16MB，是最多16MB。实际存储还是连续的，比如你可以追加一条1KB的日志，不然空间利用也太浪费了。

的确，通过分组传输是一个常见的计算机领域的办法。